DB Name	Query	Hit Count	Set Name
USPT	l19 and (knowledge near1 base)	4	L20
USPT	l17 and (data near1 center)	50	L19
USPT	l17 and (data near3 center)	128	L18
USPT	(financial near3 institution)	1781	L17
USPT	l15 and (data near3 center)	0	L16
USPT	l12 and (determin$3 near3 expansion)	1	L15
USPT	l12 and (determin$3 near3 expansionaquisition)	0	L14
USPT	l12 and (determin$3 near3 aquisition)	0	L13
USPT	l11 and (track$3 or determin$3 near3 cost)	380	L12
USPT	l9 and expert near1 system	961	L11
USPT	l9 and expert near3 system	995	L10
USPT	(knowledge near3 base)	4230	L9
USPT	l7 and (knowledge near3 base)	0	L8
USPT	l5 and (artificial near3 intelligence or expert near3 system)	10	L7
USPT	l5 and (artificial near1 intelligence or expert near1 system)	9	L6
USPT	l3 and l4	243	L5
USPT	(705/$).ccls	8204	L4
USPT	(714/$).ccls	19434	L3
USPT	l1 and knowledge	1	L2
USPT	5991791.pn.	1	L1

WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences

Search Results -

Terms	Documents
l19 and (knowledge near1 base)	4

Database:
- US Patents Full-Text Database
- US Pre-Grant Publication Full-Text Database
- JPO Abstracts Database
- EPO Abstracts Database
- Derwent World Patents Index
- IBM Technical Disclosure Bulletins

Refine Search: `l19 and (knowledge near1 base)` Clear

Search History

Today's Date: 6/27/2001

WEST

Generate Collection

Search Results - Record(s) 1 through 4 of 4 returned.

☐ 1. Document ID: US 6230197 B1

L20: Entry 1 of 4 File: USPT May 8, 2001

US-PAT-NO: 6230197
DOCUMENT-IDENTIFIER: US 6230197 B1
TITLE: Method and apparatus for rules-based storage and retrieval of multimedia interactions within a communication center

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc | Image

☐ 2. Document ID: US 6170011 B1

L20: Entry 2 of 4 File: USPT Jan 2, 2001

US-PAT-NO: 6170011
DOCUMENT-IDENTIFIER: US 6170011 B1
TITLE: Method and apparatus for determining and initiating interaction directionality within a multimedia communication center

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc | Image

☐ 3. Document ID: US 6167395 A

L20: Entry 3 of 4 File: USPT Dec 26, 2000

US-PAT-NO: 6167395
DOCUMENT-IDENTIFIER: US 6167395 A
TITLE: Method and apparatus for creating specialized multimedia threads in a multimedia communication center

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc | Image

☐ 4. Document ID: US 6138139 A

L20: Entry 4 of 4 File: USPT Oct 24, 2000

US-PAT-NO: 6138139
DOCUMENT-IDENTIFIER: US 6138139 A
TITLE: Method and apparatus for supporting diverse interaction paths within a multimedia communication center

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWMC | Draw Desc | Image

Generate Collection

Terms	Documents
l19 and (knowledge near1 base)	4

Display 10 Documents, starting with Document: 4

Display Format: TI Change Format

WEST

End of Result Set

Generate Collection

L15: Entry 1 of 1 File: USPT May 27, 1986

US-PAT-NO: 4591983
DOCUMENT-IDENTIFIER: US 4591983 A

TITLE: Hierarchical knowledge system

DATE-ISSUED: May 27, 1986

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bennett; James S.	Palo Alto	CA	N/A	N/A
Lark; Jay S.	Palo Alto	CA	N/A	N/A

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Teknowledge, Inc.	Palo Alto	CA	N/A	N/A	02

APPL-NO: 6/ 628817
DATE FILED: July 9, 1984

INT-CL: [4] G06F 15/24
US-CL-ISSUED: 364/403; 364/468, 364/478, 235/385, 29/703
US-CL-CURRENT: 706/53; 235/385, 29/703, 700/103, 700/104, 705/29, 706/59, 706/904
FIELD-OF-SEARCH: 364/400-401, 364/403, 364/468-469, 364/478, 364/513, 364/518, 209/1-2, 209/546, 209/552, 235/385, 29/33K, 29/33R, 29/4R, 29/4M, 29/428-431, 29/469, 29/564, 29/568, 29/700-703, 29/711

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4203204	May 1980	Murphy	29/703
☐	4310964	January 1982	Murphy	29/469
☐	4332012	May 1982	Sekine et al.	364/468
☐	4472783	September 1984	Johnstone et al.	364/478 X
☐	4484289	November 1984	Hemond	364/478
☐	4504919	March 1985	Fujii et al.	364/478
☐	4509123	April 1985	Vereen	364/403 X

OTHER PUBLICATIONS

James Bennett et al., "SACON: A Knowledge-Based Consultant for Structured Analysis," Stanford University Rep. STAN-CS-78-688 (Sep. 1978).
Bennett & Engelmore, "SACON: A Knowledge-Based Consultant for Structured

Analysis," Proc. of the Sixth Int. Joint Conf. on Artificial Intelligence, Tokyo (Aug. 20-23, 1979) pp. 47-49.
John McDermott, "R1: A Rule-Based Configurer of Computer Systems," Carnegie-Mellon Univ., Rep. CMU-CS-80-119 (Apr. 1980).
W. van Melle et al., The Emycin Manual, Stanford University, Rep. STAN-CS-81-855 (Oct. 1981).
Barr & Feigenbaum (eds.), The Handbook of Artificial Intelligence, William Kaufmann, Inc. (1982) vol. II, pp. 79-86, 150-154, vol. III, pp. 515-530, 541-556.

ART-UNIT: 236
PRIMARY-EXAMINER: Harkcom; Gary V.
ATTY-AGENT-FIRM: Leydig, Voit & Mayer

ABSTRACT:

A knowledge system has a hierarchical knowledge base comprising a functional decomposition of a set of elements into subsets over a plurality of hierarchical levels, a plurality of predefined functions or conditions of the elements within the subsets of a plurality of the hierarchical levels, and a predefined set of operations to perform on a user-defined set of elements responsive to the functional knowledge base. Preferably, the knowledge base is defined declaratively by assigning parent sets to offspring subsets to define the hierarchy, by indicating the conditions of the subsets which satisfy the predefined functions and by writing task blocks in an imperative language defining the sequence of operations to perform on the user-defined set of elements. Preferably the operations include matching, configuring and expanding the user-defined set of elements into the defined subsets of individual elements and evaluating the predefined functions, and the operations are executed recursively. In a specific embodiment the elements are available components for a system or item of manufacture, and the subsets of elements are sub-assemblies or functionally related components. The predefined functions define condition-action constraints to insure that the sub-assemblies have compatible components. Such a knowledge system has general applicability, is easy to maintain and incrementally modify, has transparent representation of the functional decomposition and the configuration operations, and provides explanation for an assessment of the configuration.

70 Claims, 13 Drawing figures

WEST

End of Result Set

Generate Collection

L20: Entry 4 of 4 File: USPT Oct 24, 2000

US-PAT-NO: 6138139
DOCUMENT-IDENTIFIER: US 6138139 A

TITLE: Method and apparatus for supporting diverse interaction paths within a multimedia communication center

DATE-ISSUED: October 24, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Beck; Christopher Clemmett Macleod	Oceanside	CA	N/A	N/A
Berke; Jonathan Michael	San Diego	CA	N/A	N/A
Johnstone; Joel A	San Diego	CA	N/A	N/A
Mitchell; Robin Marie	Cardiff	CA	N/A	N/A
Powers; James Karl	Carlsbad	CA	N/A	N/A
Sidell; Mark Franklin	Chapel Hill	NC	N/A	N/A
Knuff; Charles Dazler	Carlsbad	CA	N/A	N/A

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Genesys Telecommunications Laboraties, Inc.	San Francisco	CA	N/A	N/A	02

APPL-NO: 9/ 182937
DATE FILED: October 29, 1998

PARENT-CASE:
CROSS-REFERENCE TO RELATED DOCUMENTS The present application is a continuation-in-part (CIP) of several copending patent applications, the data for which is not available at the time of this writing.

INT-CL: [7] G06F 15/16
US-CL-ISSUED: 709/202; 705/1, 705/26, 705/27, 705/28
US-CL-CURRENT: 709/202; 705/1, 705/26, 705/27, 705/28
FIELD-OF-SEARCH: 709/202, 705/1, 705/26, 705/27, 705/28

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5533103	July 1996	Peavey et al.	379/96
☐	5754655	May 1998	Hughes et al.	380/24
☐	5889863	March 1999	Weber	380/25

ART-UNIT: 278
PRIMARY-EXAMINER: Lim; Krisna
ATTY-AGENT-FIRM: Boys; Donald R. Central Coast Patent Agency

ABSTRACT:

A programmable diverse interaction code module (DIM) in an enterprise-hosted multimedia call center (MMCC) for facilitates and monitors diverse interactions between parties communicating through the MMCC, and includes a database interface for access to an MMCC data repository; and an association facility for associating parties to transactions with agents and projects. The association facility assigns association identifiers to parties according to defined projects and issues, and the database interface stores transactions in the data repository. Parties to transactions include customers and business associates remote from the MMCC and agents and knowledge workers local to the MMCC, and transactions are supported in any combination between any parties. The database interface stores transactions on threaded strings, the threading associated by one or more of project, issue, and chronology, and the DIM is programmable to apply to a single enterprise project. Transactions are displayable in some cases on threaded strings, and lower-order transactions may be hidden on some strings. Methods for practicing the invention re taught as well.

20 Claims, 15 Drawing figures

End of Result Set

Generate Collection

L15: Entry 1 of 1 File: USPT May 27, 1986

DOCUMENT-IDENTIFIER: US 4591983 A
TITLE: Hierarchical knowledge system

ABPL:
A knowledge system has a hierarchical knowledge base comprising a functional decomposition of a set of elements into subsets over a plurality of hierarchical levels, a plurality of predefined functions or conditions of the elements within the subsets of a plurality of the hierarchical levels, and a predefined set of operations to perform on a user-defined set of elements responsive to the functional knowledge base. Preferably, the knowledge base is defined declaratively by assigning parent sets to offspring subsets to define the hierarchy, by indicating the conditions of the subsets which satisfy the predefined functions and by writing task blocks in an imperative language defining the sequence of operations to perform on the user-defined set of elements. Preferably the operations include matching, configuring and expanding the user-defined set of elements into the defined subsets of individual elements and evaluating the predefined functions, and the operations are executed recursively. In a specific embodiment the elements are available components for a system or item of manufacture, and the subsets of elements are sub-assemblies or functionally related components. The predefined functions define condition-action constraints to insure that the sub-assemblies have compatible components. Such a knowledge system has general applicability, is easy to maintain and incrementally modify, has transparent representation of the functional decomposition and the configuration operations, and provides explanation for an assessment of the configuration.

BSPR:
In order to provide a maintainable computer system for running expansion and validation of an order, knowledge-based systems have been considered wherein the knowledge of what the constraints are and when the constraints are to be applied is encoded declaratively in a knowledge base. A separate knowledge base interpreter interprets the knowledge base to apply the constraints. Consequently, the knowledge base interpreter need not be modified when the knowledge in the knowledge base is updated or changed. Knowledge systems have been used in general for problems that require diagnosis, recommendation, selection or classification. These problems have traditionally been performed by human experts and are not easily implemented using conventional programming techniques.

BSPR:
EMYCIN is specifically designed as a domain-independent system for constructing rule-based consultant expert system programs. Domain knowledge is represented in EMYCIN systems primarily as condition-action production rules which are applied according to a goal-directed backward-chaining control procedure. Rules and consultation data are permitted to have associated measures of certainty, and incomplete data entry is allowed. The EMYCIN system includes an explanation facility displaying the line of reasoning followed by the consultation program, and answers questions from the client about the content of its knowledge base. To aid the system designer in producing a knowledge base for a specific domain, EMYCIN provides a terse and stylized language for writing rules; extensive checks to catch common user errors, such as misspellings; and methods for handling all necessary indexing chores.

BSPR:
In addition to production rules, the knowledge base for an EMYCIN system includes a hierarchical structure called a "context tree." The elemental representation of

an object or idea is defined as a context-parameter-value triple. The context refers generally to an instance of a particular context type, the parameter refers to an attribute of the context instance and the value refers to the particular value of the parameter for the particular context instance. The context tree is defined by parent and offspring declarations for the context types.

BSPR:
Another object of the invention is to provide an intelligible knowledge base representation for hierarchical assemblies and their functionality.

BSPR:
Still another object of the invention is to provide an intelligible knowledge base representation for configuration strategies and actions.

BSPR:
Briefly, in accordance with the broadest aspect of the invention, a knowledge system for generalized representation and processing of hierarchical assemblies has a hierarchical knowledge base comprising a decomposition of a set of elements into subsets over a plurality of hierarchical levels, a plurality of respective predefined functions or conditions of the elements within the subsets at a plurality of the hierarchical levels, and a predefined set of operations to perform on a user-defined set of elements responsive to the knowledge base. For ease of maintenance and extensibility, the knowledge base is defined declaratively by assigning parent sets to offspring subsets to define the hierarchy, by indicating the conditions of the subsets which satisfy the predefined functions, and by writing task blocks in an imperative language defining the sequence of operations to perform on the user-defined set of elements. Preferably, the operations include operations for matching, configuring and expanding the user-defined set of elements into the defined subsets of individual elements and for evaluating the predefined functions, and the operations are executed recursively.

BSPR:
To provide transparent representation of the control knowledge as well as factual knowledge, the knowledge base is preferably organized into distinct portions including the assembling constraints, task blocks encoding the control strategy knowledge, functional hierarchy classes which become instantiated into bins, bin variables which take on values describing the bins and their contents, the catalog of parts, rules for expanding sub-assemblies into parts, and user-defined functions for computing the values of bin variables. The assembly constraints may be defined in terms of specified bin variables and certain built-in functions responsive to the number of specified parts in specified bins.

DRPR:
FIG. 2 is a block diagram of an exemplary M1234 computer system which is configured by the knowledge system in FIG. 1 using the knowledge base shown in Appendices II (A)-(E) to the present specification;

DRPR:
FIG. 13 is a flowchart of a DETERMINE subroutine for determining values for bin variables associated with the current bin by applying knowledge base functions associated with the current bin.

DEPR:
Turning now to FIG. 1, there is shown a block diagram generally designated 10 of a knowledge system for processing an initial configuration of elements or an order 11 to arrive at a final configuration or production request 12. The system 10 is recognized as a knowledge system since it has a domain-independent knowledge base interpreter 13 executing a built-in control procedure for interpreting a domain-dependent knowledge base 14. The knowledge base 14 encodes a generic configuration in a highly structured and transparent format. In general terms, the knowledge base interpreter 13 matches the elements of the initial configuration in the order 11 to the generic configuration in the knowledge base 14 to structure or configure the elements according to the generic configuration. The elements structured in terms of the generic configuration are stored in a working configuration memory 15.

DEPR:

Once the elements or parts of the order 11 are structured according to the generic configuration, the knowledge base intepreter 13 may apply functions or constraints 16 stored in the knowledge base 14 to the working configuration in the memory 15. The constraints 16, for example, indicate particular aspects of the working configuration 15 that are not apparent from the initial configuration or order 11. The constraints 16 may also indicate desirable changes to make to the working configuration 15. The knowledge base interpreter 13 may execute these indicated changes in order to change the working configuration. At the end of processing by the knowledge base interpreter 13, these executed changes are reflected in the final configuration 12.

DEPR:
The knowledge system 10 has a trace memory 17 for keeping a precise record of the steps performed during the matching of the elements of the initial configuration 11 to the generic configuration in the knowledge base 14 and during the application of the constraints 16 to the working configuration 15. To enable a user or order analyst 18 to inspect relevant portions of the trace 17, a user interface and explanation facility 19 is provided. The user 18 may, for example, request a complete typescript of the contents of the trace memory 17 in an easily understandable form. An example typescript is included in Appendix I at the end of the present specification.

DEPR:
The knowledge base 14 is arranged so that it is easily modified and maintained to reflect desired changes in the generic configuration. The modification and maintenance is performed by a knowledge engineer 20 and the knowledge base 14 is accessed through an editor 21 called a knowledge base maintenance facility. The knowledge engineer 20 initially creates the knowledge base 14 collecting and encoding knowledge about the generic configuration according to a precise format and syntax so that the knowledge is transparent to the knowledge engineer and is machine readable by the knowledge base intepreter 13. For the processing of an order of parts 11 to generate a production request 12 for a system or product, the knowledge engineer 20 collects engineering and marketing data on the generic configuration of the product and its constraints and encodes the information according to the syntax recognized by the knowledge base interpreter 13.

DEPR:
The specific format of the knowledge base 14 is chosen so that the knowledge base can be easily updated to reflect the addition of new products and changes in current products. The generic configuration is represented as a functional hierarchy 22 which describes the product in various degrees of particularity. When a product is modified, only the particular portions of the functional hierarchy 22 need be changed. Moreover, since the hierarchy is in terms of the function of performance of parts of the generic configuration for the product, the changes required in the functional hierarchy 22 are minimal.

DEPR:
For each function in the functional hierarchy 22, a separate task block 23 is provided to encode corresponding steps in the control procedure followed by the knowledge base interpreter 13 to match the parts or elements of the initial configuration 11 to the functional hierarchy 22. A change in the functional hierarchy usually requires only a localized change in the respective task block 23. Also, the constraints 16 correspond to particular functions in the functional hierarchy 22 so that the constraints 16 are also easily modified. Consequently, the constraints associated with a particular function in the functional hierarchy are applied in a manner specified by the respective task block 23.

DEPR:
It should be noted that even though the functional hierarchy 22 might not change during the life of a particular product, the parts or elements making up the product may change during the life of the product. These changes are reflected in a parts catalog 24. Moreover, to simplify the process of taking an order from a customer, an order typically includes packages of parts implementing certain features. To enable the knowledge base interpreter 13 to find the particular parts included in the packages in the order 11, the knowledge base 14 includes a set of expansion rules 25.

DEPR:
In addition to providing a framework for clearly intelligible and maintainable

representation of knowledge about a product, the functional hierarchy 22 provides the structure or framework for the control procedure executed by the knowledge base interpreter 13 to match the parts in the order 11 to the generic configuration in the knowledge base 14 and to apply the constraints 16 to validate the order 11 and generate a production request 12. The working configuration memory 15 is organized into respective bins of parts 26 corresponding to the major functional components of the product defined by the functional hierarchy 22. Each bin of parts 26 corresponds to a structural or functional assembly in the product and the matching process is performed by inputting the initial configuration or list of parts 11 into an initial bin of parts 26 and sequentially matching the parts in parent bins to assembly descriptions for offspring bins and transferring the matching parts to the offspring bins. The task blocks 23 specify the particular assembly descriptions and the specific sequence of matching and transferring the parts. Before matching and transfering the parts from certain bins, however, the expansion rules 25 must sometimes be applied to "break open" certain packages of parts in the order 11 since different parts from the same packages sometimes match the assembly descriptions of different offspring bins. At the end of the transfer process, the bins of parts 26 contain respective parts from the initial configuration 11. Hence, the parts in the initial configuration 11 have been structured or configured according to the generic product configuration or functional hierarchy 22 and thus the constraints 16 may be applied to their respective bins of parts 26.

DEPR:
In order to simplify the process of applying the constraints, the constraints 16 may include bin variables defined for particular respective parent bins and which may be referenced in the parent bins and respective offspring bins of the parent bins. The knowledge base 14 includes separate declarations 28 of the bin variables. The respective task blocks 23 include steps for determining the values for the bin variables 27 and the values are stored in the working configuration memory 15. These steps in the task blocks 23 may include function calls to knowledge base functions 29 in the knowledge base 14. The knowledge base functions 29 typically count parts and perform numerical computations to determine values for bin variables. In addition to conditioning constraints, bin variables may be used to condition the sequence of execution of the steps in the task blocks in order to perform conditional expansion, matching and transfer of parts and assemblies.

DEPR:
From the block diagram in FIG. 2 and other technical information about the M1234 computer, the knowledge engineer 20 generates a functional hierarchy description of the M1234 computer as illustrated in FIG. 3. In FIG. 3 the hierarchy is arranged in the form of a tree diagram generally designated 40, having an initial or root node 41, intermediate or branch nodes 42-46, and several terminal or leaf nodes 47-60. Each node in the functional hierarchy 40 represents a single functional component. The initial node 41 and the intermediate nodes 42-46 represent functional components that are composed of a number of more specific functional components. Specifically, the functional hierarchy 40 is defined by including declarations of parent functional components in the definitions of the intermediate 42-46 and terminal 47-60 functional components. The functional component MAINFRAME 42, for example, has the parent functional component SYSTEM 41. The functional hierarchy 22 of the knowledge base 14 (FIG. 1) corresponding to the tree diagram 40 in FIG. 3 is listed in Appendix II (A). According to the particular knowledge base syntax of Appendix II (A), it is said that the declared offspring functional component "COMPOSES" its parent functional component.

DEPR:
It should be noted that the consultation typescript of Appendix I was generated by the interpretation of a knowledge base in Appendices II (A)-(E). The task blocks are in Appendix II (B); the parts catalog is in Appendix II (C); the constraints, bin variable declarations, and knowledge base functions are in Appendix II (D); and the expansion rules are in Appendix II (E).

DEPR:
In general the knowledge base is in the form of a list of declarations of task blocks which are invoked and executed when the system is run, declarations of knowledge base functions which are applied or evaluated during execution of the task blocks to set values of the bin variables, and declarations of knowledge

base objects which become instantiated to generate one or more object instances in working memory during execution of the task blocks. Declarations of task blocks and knowledge base functions are preceded by the identifiers DEFTASKBLOCK and DEFKBFUN, respectively, followed by a specified name for the task block or knowledge base function. Declarations of objects are preceded by the identifier DEFCLASS, followed by a specified name for the object and, in parentheses, the class or type of the instantiated object. Object types include BIN (an instance of component or node in the functional hierarchy), PRODUCT.ID (an instance of a particular part), BVI (a bin variable instance), CTI (an instance of applying a constraint or product expansion), and QUEUE. Respective QUEUE instances keep track of the current step of execution in each task block so that the execution of one task block may be interrupted to execute another task block.

DEPR:
The declarations or definitions of object types in the knowledge base include values for instance variables or IVARs associated with instances of the object types. For each offspring BIN, for example, the IVAR "COMPOSES" is set to the name of the BIN's parent BIN. Each BIN and PRODUCT.ID has an IVAR "DESCR" including a description of the respective function or part in English so that the user interface and translation facility 19 may generate comprehensible explanations and traces of the operation of the system 10 (FIG. 1).

DEPR:
It is apparent that the functional hierarchy serves both to organize the configuration process and to guide the knowledge engineer in designing and maintaining the knowledge base. By understanding the structure of the functional hierarchy and the effects of the configuration strategy used to process an order, the knowledge engineer can understand how the system will configure an order and how the application of constraints, product definitions, and expansion will affect the processing of that order.

DEPR:
The knowledge system 10 of FIG. 1 has its greatest utility for configuring complex systems such as the M1234 computer as described in FIGS. 2 and 3 and the typescript and knowledge base of Appendices I and II (A)-(E). For the sake of illustration, however, the internal operation of the system will be described in connection with a more easily understood minicomputer system corresponding to the functional hierarchy generally designated 70 in FIG. 4 and corresponding to the consultation typescripts and knowledge base in Appendices III (A)-(B) and IV (A)-(D).

DEPR:
In the functional hierarchy 70 of FIG. 4, each node such as the initial node 71 is labeled by its functional component and has a class name enclosed in parentheses corresponding to the classes declared in the knowledge base functional hierarchy of Appendix III. The minicomputer 71 is presumed to have two major functional components. The main-frame 72 of the minicomputer encompasses all of the active components of the computer such as the central processing unit 73, a computer memory 74, and input/output devices 75. These active devices 73-75 are physically tied together by a card cage 76. The card cage 76 includes mechanical support for various circuit boards performing the active functions of the computer and also provides a network of electrical connections for the transfer of data among the circuit boards. The second functional component of the minicomputer 71 is a power supply 77 which feeds power to the circuit boards through power supply lines running through the card cage 76.

DEPR:
Once the order is received by the knowledge system 10, the parts in the order are checked for consistency by applying the constraints in the knowledge base (Appendix IV (D)). In general, constraints specify engineering or marketing conditions which must be satisfied for the order lines. The constraints also specify the particular action that should be taken in response to whether the conditions are satisfied when the constraints are applied. For the MINI-1000 minicomputer, the order must include at least one RAM-1000 random access memory board, at least one ROM-1000 read only memory board, at least one TERMINAL-1000 computer terminal, and at least one IF-1000 interface board. The action to take when any of these parts is found to be missing is to add one of the respective missing parts, since these parts are essential to the operation of the minicomputer. The minicomputer also has a number of essential components that

must be added if missing, but which can only appear with a quantity of one. The minicomputer must have one and only one power supply (SUPPLY-05, SUPPLY-10, or SUPPLY-15), one and only one CPU-1000 central processing unit card, and one and only one CAGE-1000 card cage.

DEPR:
The minicomputer 71 has a global power constraint due to the fact that the customer may order a power supply that is smaller than that required under the worst case conditions. The power required is a function not only of the number of cards, but of the particular cards that are used. The CPU-1000 card requires two amperes of current, the RAM-1000 card requires one ampere of current, the ROM-1000 card requires 0.5 amperes of current, and the IF-1000 interface card requires 0.2 amperes of current. The knowledge base in Appendix IV (D) includes a special knowledge base function called "COMPUTE.POWER.NEEDED" for calculating the total amount of power required. Based on the required power, a five ampere power supply SUPPLY-05, a ten ampere power supply SUPPLY-10, or a fifteen ampere SUPPLY-15 should be included in the order. The knowledge base in Appendix IV (D) includes a set of modification constraints to ensure that the smallest power supply is included consistent with the total power requirement.

DEPR:
For the MINI-1000 minicomputer, the computer's functional hierarchy, parts catalog, and constraints are relatively simple so that the order checking process in FIG. 5 could easily be implemented using a conventional programming language such as the BASIC language. Since an order for a MINI-1000 computer is merely a list of the indivisible components of the computer, the constraints are easily applied. More importantly, the functional class which each part implements is readily apparent from the type of part. For the M1234 computer described in the knowledge base of Appendices II (A)-(E), however, it would be very difficult to use a conventional programming language to implement an order checking computer system and the computer system would be very difficult to maintain.

DEPR:
By using the techniques already described, the components and functionality of a complex product such as a computer can be transparently and explicity represented in the knowledge base. In accordance with another aspect of the present invention, the definition of configuration constraints is clearly separated from the configuration strategies and actions. In particular, a built-in control procedure is provided which permits the constraints to be defined for particular functional components and separately determines which parts are applicable to the respective constraints. At the most basic level, this separation requires a process which builds an explicit representation of the assemblies of the product from the parts in the order. Shown in FIG. 6, for example, is an explicit representation generally designated 90 for the MINI-1000 minicomputer and the example order given above. The explicit representation 90 is a tree of bins corresponding to the functional hierarchy of FIG. 4. In FIG. 6 the name of each bin corresponds to the respective functional component or bin class in the hierarchy of FIG. 4. Moreover, the particular instance of the class is identified by a numeral enclosed in parentheses. Although the examples in the appendices only use a single instance of each functional class, the knowledge engineer may write task blocks which create a specified number of bins for a specified functional class.

DEPR:
It should be noted that the knowledge base interpreter 13 is always working on a particular bin at any given time. The name of this particular bin is stored in a global variable called "CURRENT.BIN". In step 103 this global variable "CURRENT.BIN" is set to the value "UNASSIGNED" to record that the configuration process is starting in the "UNASSIGNED" bin. In step 104 "UNASSIGNED" task block is found, and it is excecuted in step 105 so that the rest of the configuration process is controlled solely by task blocks.

DEPR:
Translated into English, the UNASSIGNED task block above first inputs the order of parts. Then it determines the current date, the ordered parts or order lines of the order, and the product line corresponding to the ordered parts. It is presumed that the knowledge system is currently checking the order. Thus, the task block for the ordered product line is fetched from the knowledge base. This task block is, for example, the MINICOMPUTER task block in Appendix IVB. Then a

FIND operation is performed to obtain a list of all the parts in the order lines, and the CONFIGURE operation is performed to create the initial bin in the bin tree, to transfer all the parts in the order lines into the initial bin, and to temporarily transfer execution to the MINICOMPUTER task block. When execution returns, the parts are configured in the bins as shown in the bin tree 90 of FIG. 6. A final set of EMPTY.BIN constraints are applied to ensure that all of the order lines in the order have been properly configured. If so, the FINAL.PARTS are determined and a production request is generated by the OUTPUT.ORDER statement.

DEPR:
The UNASSIGNED task block as well as the task blocks in the knowledge base include a mixture of LISP code written by the knowledge engineer, built-in functions supplied by the LISP compiler, and special functions or subroutines comprising the built-in control procedure of the knowledge base interpreter 13 (FIG. 1). The primary built-in subroutines include the EXECUTE subroutine already described in conjunction with FIG. 8, a FIND subroutine shown in FIG. 9, a CONFIGURE subroutine shown in FIG. 10, a CHECK subroutine shown in FIG. 11, an EXPAND subroutine shown in FIG. 12, and a DETERMINE subroutine shown in FIG. 13.

DEPR:
Shown in FIG. 11 is a flowchart of the check subroutine 140. In the first step 141 the knowledge base is searched to obtain a list (CL) of the constraints applicable to the functional class of the current bin and which satisfy the given constraint description. Then, in step 142 a pointer (P3) is set to the first constraint in the constraint list (CL). Next, in step 143 the condition portion of the constraint in the list pointed to by the pointer (P3) is evaluated. The condition of the constraint is typically a Boolean function of the conditions of the various parts in the bin. If the condition of the constraint is satisfied, as determined by testing whether the Boolean value of the conditions is true, then in step 145, the action specified by the constraint is executed. Otherwise, the action specified by the constraint is not executed. In step 146 the pointer (P3) is advanced in order to point to the next constraint in the constraint list (CL). The pointer (P3) must be checked, however, in step 147 to determine whether all of the constraints in the list have been applied. If the pointer (P3) is still within the range of the constraint list (CL), then execution jumps to step 143 to apply the next constraint. Otherwise, the CHECK subroutine 140 is finished and execution returns to the current task block.

DEPR:
It should be noted that during certain steps in the sorting or configuring process, certain kits or groups of parts must be expanded into their component parts. An EXPAND operation is provided for expanding the parts in the current bin into their component parts by applying predefined expansion rules associated with the current bin. An EXPAND subroutine generally designated 150 is shown in FIG. 12. In the first step 151, a list (EXL) of all of the expansion rules that apply to the parts in the current bin are obtained from the knowledge base. Next, in step 152 a pointer (P4) is set to the first expansion rule (EX) in the expansion rule list (EXL). Then in step 153 the particular expansion rule (EX) pointed to by the pointer (P4) is obtained. The expansion rule (EX) is checked in step 154 to determine whether it has already been marked expanded. It should be noted that a particular expansion rule may have already been applied since a particular expansion rule may apply to more than one bin. If the expansion rule has not already been applied, then in step 154' the expansion is applied by adding to the current bin the parts included in the composing assembly description of the expansion rule. The composing assembly description is a list of the parts included in the respective kit for the respective expansion rule. Once the expansion rule has been applied, it is marked expanded in step 155. Then, in step 156 the pointer (P4) is advanced in order to point to the next expansion rule in the list. However, in step 157 the value of the pointer (P4) must be checked to determine whether it does point to another expansion rule in the list, or whether all of the expansion rules in the list have been applied. If the point (P4) points to another expansion rule in the list, then execution jumps to step 153. Otherwise, execution returns to the current task block.

DEPR:
During the execution of task blocks and the application of constraints, it is in many cases desirable to condition the execution of a particular task block step upon the conditions or attributes of the parts included in the bin for the task

block. As noted above, bin variables are provided for this purpose and the bin variables may also be included in the condition portions of constraints. A DETERMINE operation is provided for specifying when to determine values for the bin variable instances associated with the current bin by applying the knowledge base functions associated with the current bin. Shown in FIG. 13 is a flowchart of a DETERMINE subroutine generally designated 160. In the first step 161 a list (BVL) of bin variables for the current bin are obtained from the knowledge base. Next, in step 162 a pointer (P5) is set to the first bin variable in the bin variable list (BVL). Then in step 163, the particular bin variable (BV) pointed to by the pointer (P5) is obtained. Before determining a value for the bin variable instance (BV), in step 164 the bin variable instance is inspected to determine whether it already has a value. If not, then in step 165 a list (KBFL) of knowledge base functions concluding a value for the bin variable (BV) is obtained from the knowledge base. Then in step 166 another pointer (P6) is set to the first knowledge base function in the function list (KBFL). In step 167 the particular knowledge base function (KBF) pointed to by the pointer (P6) is obtained. This knowledge base function (KBF) is evaluated in step 168 to obtain a respective value (V). This respective value (V) might not, however, be a legal value for the particular bin variable (BV). In step 169 the declared legal values for the particular bin variable (BV) are obtained from the knowledge base and compared to the particular value (V) to determine whether the value is a legal value for the bin variable (BV). If not, then in step 70 the knowledge base function pointer (P6) is advanced. In step 171 the pointer is compared to the end of the range for the function list (KBFL) to determine whether the pointer (P6) is pointing to another knowledge base function. If so, then execution may jump back to step 167 in an attempt to apply this other knowledge base function. Otherwise, there are no more knowledge base functions to try so that in step 172, a value of "NO VALUE" is assigned to the bin variable instance (BV).

DEPR:
Once the knowledge engineer encodes a specific configuration strategy into the knowledge base, a test configuration can be run using a "TRACE" option so that the user interface and explanation facility (19 FIG. 1) generates a comprehensible translation of the configuration strategy. For the MINI-1000 minicomputer, for example, the trace in the typescript of Appendix III (A) explains in detail the specific configuration strategy that is followed. This explanation includes the specific task block statements that are executed and any action taken upon application of the constraints. The configuration strategy for the MINI-1000 minicomputer order in Appendix II (A) first looks at the set of ordered parts to determine that the product line is a MINI-1000 minicomputer. These parts are dumped into the root bin MINI(1) in FIG. 6. Then the parts implementing the MAIN-FRAME function are found and the MAIN-FRAME(1) bin is created and these parts are transferred to the MAIN-FRAME(1) bin. In recursive fashion, the parts in the MAIN-FRAME(1) bin are found which implement the CPU function. A CPU(1) bin is created and these parts are transferred to the CPU(1) bin. The requirements for the CPU(1) bin are checked and it is found that a required CPU board is missing. The knowledge system 10 asks the user for permission to add one CPU-1000 circuit board, thereby completing the CPU(1) bin. Execution returns to the MAIN-FRAME task block. The components implementing the MEMORY function are found and a new MEMORY(1) bin is created to receive these parts. In recursive fashion, the parts in the MEMORY(1) bin implementing the RAM function are found and a new RAM(1) is created to receive these parts. The contents of the RAM(1) bin are checked to ensure that at least one RAM-1000 circuit board is included. Execution then returns to the MEMORY task block which finds the parts implementing the ROM function and creates a new ROM(1) bin to receive these parts. The contents of the ROM(1) bin are checked to ensure that at least one ROM-1000 circuit board is included. Then, execution returns to the MEMORY task block which checks the overall requirements for the MEMORY function. In particular, a MEMORY.CT1 constraint is applied which counts the total number of RAM and ROM boards to ensure that the total number does not exceed seven. When this constraint is applied, it is found that the total number does not exceed seven. Execution then returns to the MAIN-FRAME task block. Next, the MAIN-FRAME task block finds all of the parts in the MAIN-FRAME(1) bin which implement the I/O function, creates a new I/O(1) bin and transfers these parts to the new bin. Execution passes to the I/O task block which finds all of the components in the I/O(1) bin implementing the TERMINAL function, and a new TERMINAL(1) bin is created to receive these parts. Upon execution of the TERMINAL task block, the requirements or constraints for the TERMINAL function are applied. It is found that there is at least one terminal in the TERMINAL(1) bin. It is also found that

the number of terminals does not exceed four. Execution then returns to the I/O task block.

DEPR:
Inspection of the constraints, bin variable declarations and knowledge base functions for the M1234 computer and the MINI-1000 minicomputer knowledge bases in Appendices II (D) and IV (D) reveal that there are many built-in functions available to the knowledge engineer which simplify the knowledge engineer's task of creating the task blocks, constraints, and knowledge base functions. Moreover, the knowledge base is organized so that the knowledge engineer has ready access to the conditions of the parts upon which the constraints and knowledge base functions are responsive and can manipulate the working configuration in any desired fashion.

DEPR:
Many of the built-in functions recognized by the knowledge base interpreter 13 of the knowledge system 10 (FIG. 1) have particular utility in conjunction with constraints. Constraints always apply to a specified set of parts. The parts are specified by a list of BIN.TYPES, which are the FUNCTION.CLASS entries in the functional hierarchy that the constraint applies to. The constraint also includes a STATEMENT of conditions to test against the parts in those bins. The STATEMENTs often test relationships between values of different bin variables, which must be declared in the BIN.VARS.REQUIRED field of the constraint declaration. Also, a list of CHECK.KEYWORDS may be specified to control when certain constraints are applied. These keywords are used by the CHECK operations in the task block steps to select relevant subsets of constraints to apply at different times in a task block.

DEPR:
The knowledge system provides a number of important specializations of the CONSTRAINT knowledge base type, which are further described below. All product type and specific constraints are defined in terms of these specializations. The system also considers product expansions as a particular kind of CONSTRAINT since they add parts to a bin if certain conditions, such as whether a particular product was ordered, are met. In general, the types of constraints include constraints on specific parts, warning constraints, and modification constraints.

DEPR:
As noted above, an expansion rule is a kind of constraint that applies to specific parts. Expansion rules are specified by EXPANSION.CT objects in the knowledge base. The FROM field of an EXPANSION.CT specifies the composite part to be expanded; the TO field specifies the list of component parts to add to the bin; the WHEN field specifies the conditions under which this expansion should occur; and the MESSAGE field specifies the messages that should be printed when the expansion succeeds or fails.

DEPR:
Modification constraints are defined in the knowledge base either as MODIFY-ON-SUCCESS.CT objects or a MODIFY-ON-FAILURE.CT objects. A warning message is given to the user and the contents of the current bin are modified if the conditions in the STATEMENT field succeed or fail, respectively. The knowledge engineer may specify a MESSAGE that will warn the user that a modification must be made. The modifications are specified by providing values for one or more of the fields SET-PARTS, ADD-PARTS, and DEL-PARTS. If a list of parts is specified for the field SET-PARTS, the current bin is emptied and the list of parts (in the quantities specified) is placed in the bin. Then, if a list of parts is specified in the ADD-PARTS field, new parts are added to the bin in the quantities specified. If a list of parts is provided in the DEL-PARTS field, existing parts are deleted from the bin in the quantities specified.

DEPR:
In the knowledge base, the BIN.VAR declarations must include a LEGAL.VALUE field indicating the legal value for the bin variable (e.g., integer, string, etc.), a HOW.TO.DETERMINE field including a list of KBFUNCTIONs that can be used to determine the value of the bin variable, and a BIN.TYPES field including the bin types or function classes with which the variable is associated. It should be noted that values of BIN.VAR instances can be referenced by constraints applied by task blocks that are lower in the bin tree than where the BIN.VAR instance is

created. Thus, the BIN.TYPES should associate the bin variable with bins high enough in the bin tree so that the bin variables can be referenced by all of the appropriate contraints.

DEPR:
It should be noted that the knowledge base itself may be divided into sections so that the sections my be identified and referenced by the task blocks and constraints. The name for each portion of the knowledge base is preferably composed of a knowledge base prefix and a unique suffix identifying the particular portion or file of the knowlege base. In the preferred embodiment of Appendix II (A)-(E) the suffix FH identifies the file containing the functional hierarchy and the product line task blocks. The suffix NCT identifies the file containing the product line constraints, the bin variables, and the knowledge base functions. The suffix BX contains product line descriptions. The suffix AK identifies the file containing product line packages. The suffix FT identifies the file containing product line features. The suffix KT identifies the product line kits. The suffix KV identifies the file containing the product line kit versions. The suffix XT identifies a file containing miscellaneous parts. The suffix EX identifies the file containing product expansions. A knowledge base file with the suffix MODS is used to record all modifications or changes made to the knowledge base since the last modification by the knowledge engineer using the knowledge base maintenance facility 21 (FIG. 1).

DEPR:
The knowledge base maintenance facility 21 preferably has a number of commands used by the knowledge engineer to create, maintain, and augment the knowledge base 14. These commands include, for example, eight commands for knowledge base data and control functions, three commands for knowledge base editing functions, and four commands for knowledge base file maintenance.

DEPR:
The eight knowledge base data and control commands include SELECT, PP, CHANGES, CONTENTS, CONTROL, IGNORE.CHANGES, REMEMBER.CHANGES, and LIST.KBSETS.

DEPR:
The command SELECT selects a knowledge base to be modified and loads the knowledge base into the host computer system so that it may be further manipulated by the knowledge base maintenance facility 21.

DEPR:
The command PP displays the status of the current knowledge base, including whether changes are being recorded, whether the changes will be automatically saved, how many modifications have been made since the last time the knowledge base was saved, (i.e., written back from the working memory of the knowledge base maintenance facility 21 to the more permanent memory from which the knowledge base was loaded), and how many objects are currently in the current knowledge base.

DEPR:
The CHANGES command displays the modifications made to the knowledge base since the most recent migration, (see MIGRATE below) and indicates which objects have been added, changed, or deleted from the knowledge base.

DEPR:
The command CONTENTS displays the names of all of the objects in the current knowledge base.

DEPR:
The command CONTROL controls whether the system automatically saves modifications after a certain number of additions, edits, and deletions have been made to the current knowledge base.

DEPR:
The command IGNORE.CHANGES tells the knowledge base maintenance facility 21 to ignore any subsequent additions, modifications, or deletions.

DEPR:
A command LIST.KBSETS displays the contents of specified main knowledge base sets after asking the user to specify the names of the sets.

DEPR:
The three knowledge base editing commands include ADD, EDIT, and DELETE. The command ADD adds new objects to the current knowledge base and prompts the knowledge engineer for the knowledge base type and the name of the new knowledge base object. The command EDIT modifies existing objects in the current knowledge base and prompts for the name of the knowledge base object. The command DELETE removes objects from the current knowledge base and prompts for the name of an existing knowledge base object.

DEPR:
The four knowledge base file maintenance commands include SAVE, MIGRATE, BUILD, and CREATE. The command SAVE saves the copy of the current knowledge base changes in a modification file. The command MIGRATE distributes the current knowledge base modification into the main knowledge base files and writes out a new, empty copy of the modifications file. The command BUILD saves a copy of the current executable system on a file and prompts the knowledge engineer for the name of the file. The command CREATE creates a new knowledge base for a new product line, and automatically prompts for a knowledge base prefix to identify the new product line.

DEPR:
The knowledge system 10 has been described above primarily for checking and modifying the order lines in an order for a flexibly assembled product. It should be noted, however, that the modification constraints in part redesign the ordered product. The system 10 is equally useful as a design tool. The design procedure is illustrated by the typescript in Appendix III (B) wherein an order for a MINI-1000 minicomputer merely includes the model number of the desired product. The set of modification constraints in the knowledge base for the MINI-1000 product line (Appendices IV (A)-(D)) designs a minimal system consistent with the modification constraints including one CPU-1000 card, one RAM-1000 card, one ROM-1000 card, one TERMINAL-1000 computer terminal, one IF-1000 card, one CAGE-1000 card cage, and one SUPPLY-05 five ampere power supply.

DEPR:
It should also be noted that the knowledge engineer has great flexibility in defining the control procedure for the knowledge base interpreter 13. The knowledge engineer, for example, may write specialized knowledge base functions and task blocks to perform specialized configuration operations. The knowledge engineer may also modify the built-in control procedure of the knowledge base interpreter 13 to modify the control procedure in any desired fashion.

DEPR:
In view of the above, a knowledge system has been described for generalized representation and processing of hierarchical assemblies. The configuration strategies and actions are defined in task blocks which are completely separate from the functional hierarchy which describes the generic configuration of the product and the parts catalog which defines the individual elements available for configuration. Moreover, the constraints are completely separate from the task blocks, functional hierarchy, and parts catalog so that the definition of configuration constraints are clearly separated from configuration checking strategies and actions. The knowledge base interpreter has a built-in control procedure and built-in functions which enable the knowledge engineer to design and implement readily a desired configuration checking strategy. Since the imperative language of the task blocks clearly defines the configuration checking strategies and actions, a trace of a configuration operation is easily stored in a trace memory including both the actual steps executed and the resulting actions such as warnings or modifications to the configuration. Thus, an explanation facility can be provided which generates an intelligible and comprehensible explanation of the configuration based on the record in the trace memory.

DEPV:
KNOWLEDGE BASE FOR M1234 COMPUTER FUNCTIONAL HIERARCHY

DEPV:
KNOWLEDGE BASE FOR M1234 COMPUTER TASK BLOCKS

DEPV:
KNOWLEDGE BASE FOR M1234 COMPUTER PARTS CATALOG

DEPV:
KNOWLEDGE BASE FOR M1234 COMPUTER CONSTRAINTS & BIN VARIABLES & KNOWLEDGE BASE FUNCTIONS

DEPV:
KNOWLEDGE BASE FOR M1234 COMPUTER EXPANSION RULES

DEPV:
KNOWLEDGE BASE FOR MINICOMPUTER FUNCTIONAL HIERARCHY

DEPV:
KNOWLEDGE BASE FOR MINICOMPUTER TASK BLOCKS

DEPV:
KNOWLEDGE BASE FOR MINICOMPUTER PARTS CATALOG

DEPV:
KNOWLEDGE BASE FOR MINICOMPUTER CONSTRAINTS & BIN VARIABLES & KNOWLEDGE BASE FUNCTIONS

DEPV:
A type of bin denoting that the configuration system has not yet determined what product the parts of the bin will implement and therefore which knowledge base should be used to configure the order.

CLPR:
1. A knowledge system comprising a computer having a memory storing a knowledge base, said knowledge base including

CLPR:
3. The knowledge system as claimed in claim 1, wherein said knowledge base further includes a catalog of components including component names and respective names of assemblies comprising said components, and wherein said descriptions of assemblies include the names of some of said assemblies in said catalog of components.

CLPR:
7. The knowledge system as claimed in claim 1, wherein the knowledge base further includes respective actions for said conditions, and wherein said control procedure includes a set of control steps for carrying out said actions for indicating whether the respective conditions are satisfied.

CLPR:
12. The knowledge system as claimed in claim 1, wherein some of said assemblies are sub-assemblies of other of said assemblies, and wherein said knowledge base includes a definition of a hierarchy of said descriptions of assemblies defining offspring-parent relationships between said descriptions of sub-assemblies and the respective descriptions of assemblies which comprise said sub-assemblies, and wherein said control procedure for matching the initial set of components to the predefined descriptions of assemblies includes means for first matching the list of components to the parent descriptions of assemblies, and then matching the list of the respecting matching components for the respective parent assemblies to their respective offspring descriptions of sub-assemblies.

CLPR:
15. The knowledge system as claimed in claim 1, wherein said knowledge base further comprises a set of expansion rules defining expandable components including some of said components in the initial set of components in terms of respective sub-components of the expandable components and wherein said control procedure further includes a set of control steps for selecting expandable components from the initial set of components and adding the respective sub-components corresponding to the selected expandable components to the initial set of components.

CLPR:
18. The knowledge system as claimed in claim 17, wherein said knowledge base further includes respective modification actions for making specified changes to the recorded set of matching components in the respective bin portions of said

working configuration portion of said memory, and wherein the said control procedure for applying the sets of conditions includes a set of control steps for conditionally executing said changes in response to whether the respective conditions are found to be satisfied for the respective assemblies when the respective conditions are applied.

CLPR:
26. The knowledge system as claimed in claim 21, wherein said conditions include Boolean functions of specified attributes of specified components, and wherein said knowledge base includes a knowledge base function associated with a respective one of said specified attributes, said knowledge base function being implicitly invoked when said condition is applied including said Boolean function of said specified attribute, so that the knowledge base function determines a value for its respective specified attribute.

CLPR:
29. A knowledge system comprising a computer having a memory storing a knowledge base, said knowledge base including

CLPR:
33. The knowledge system as claimed in claim 32, wherein the knowledge base includes definitions of variables for specifying the attributes of components included in the descriptions of respective assemblies.

CLPR:
34. The knowledge system as claimed in claim 33, wherein said knowledge base includes knowledge base functions for at least some of said variables specifying steps for determining values for the respective attributes of said components, and said control procedure includes a set of control steps for implicitly invoking and executing the respective knowledge base functions when applying constraints including conditions referencing the respective variables having the knowledge base functions.

CLPR:
37. The knowledge system as claimed in claim 35, wherein said knowledge base includes expansion rules defining sub-components for respective components, and wherein said control procedure includes a set of control steps for searching a specified bin for at least some of said components having sub-components defined by said expansion rules, and recording the respective sub-components in the specified bin.

CLPR:
43. The knowledge system as claimed in claim 42, wherein the knowledge base includes expansion rules specifying that at least some of said components are expandable and are comprised of sub-components and wherein the imperative language statements include a separate statement for specifying that for each expandable component in a specified bin, its respective sub-components are to be recorded in the specified bin.

CLPR:
47. The knowledge system as claimed in claim 40, wherein said knowledge base includes a plurality of separate portions for processing predetermined lists to configure different respective products, and wherein the control procedure includes control steps for obtaining said predetermined list by an input operation and for selecting the knowledge base portion corresponding to a product number in said predetermined list.

CLPR:
52. The knowledge system as claimed in claim 50, wherein the knowledge base includes predetermined changes to the composition of said lists of matching elements stored in said working configuration portion of said memory, and wherein said functions include respective Boolean condition functions for indicating said changes, and wherein the control procedure includes control steps for executing the indicated changes to thereby conditionally change said lists of matching elements.

CLPR:
59. The knowledge system as claimed in claim 50, wherein said knowledge base includes a plurality of task blocks specifying in part the operation of said

control procedure, and said subsets are associated with particular ones of said task blocks, and said task blocks specify operations to perform with respect to their associated subsets.

CLPR:
61. A knowledge system for checking a production request for a flexibly-assembled product, said request including a list of part names and respective quantities, said knowledge system comprising a computer having a memory storing a knowledge base, said knowledge base including

CLPR:
62. The knowledge system as claimed in claim 61 wherein the knowledge base further comprises task blocks for respective ones of said assemblies, said task blocks including imperative language statements for specifying control procedure steps responsive to the parts in the bins for the respective assemblies.

CLPR:
63. The knowledge system as claimed in claim 62 wherein the knowledge base further comprises a separate set of expansion rules specifying the composition of expandable parts in terms of sub-parts, and wherein the task blocks include imperative language statements for adding the sub-parts of expandable parts in the bins to the respective bins.

CLPR:
64. The knowledge system as claimed in claim 61, wherein said knowledge base further comprises a set of actions specifying changes to the set of parts configured into at least some of said assemblies in response to the respective conditions for said assemblies, and wherein said control procedure includes a set of control steps for changing the sets of parts configured into the respective assemblies in the fashion indicated by said actions in response to whether the respective conditions are satisfied when the respective conditions are applied.

CLPR:
67. A knowledge system for designing a product including assemblies of predetermined parts, said knowledge system comprising a computer having a portion of memory storing a knowledge base and a portion of memory subdivided into respective bins for receiving selected names of said parts for comprising the respective assemblies,

CLPR:
69. The knowledge system as claimed in claim 67 wherein the knowledge base further comprises task blocks for respective ones of said assemblies, said task blocks including imperative language statements for specifying control procedure steps responsive to the parts in the bins for the respective assemblies.

CLPV:
a knowledge base including knowledge about a set of related elements, and

CLPV:
said knowledge base including a control procedure for

CLPW:
wherein said knowledge base further includes a hierarchy defining at least some of said assemblies as offspring sub-assemblies of respective parent assemblies, and wherein said assembly constraints associated with said parent assemblies have conditions referencing the conditions of parts in the respective offspring bins, and